(12) DEMANDE INTERNATIONALE PUBLIÉE EN VERTU DU TRAITÉ DE COOPÉRATION EN MATIÈRE DE BREVETS (PCT)

(19) Organisation Mondiale de la Propriété Intellectuelle
Bureau international

(43) Date de la publication internationale
1 septembre 2005 (01.09.2005)

PCT

(10) Numéro de publication internationale
WO 2005/081141 A2

(51) Classification internationale des brevets[7] : G06F 17/50

(21) Numéro de la demande internationale :
PCT/FR2005/000124

(22) Date de dépôt international :
20 janvier 2005 (20.01.2005)

(25) Langue de dépôt : français

(26) Langue de publication : français

(30) Données relatives à la priorité :
0400488 20 janvier 2004 (20.01.2004) FR

(71) Déposants *(pour tous les États désignés sauf US)* : **CERVVAL** [FR/FR]; 120, Kervézingar Izella, F-29470 Plougastel Daoulas (FR). **ECOLE NATONALE D'INGENIEURS DE BREST** [FR/FR]; Technopôle Brest Iroise, F-29238 Brest Cedex 3 (FR).

(72) Inventeurs; et

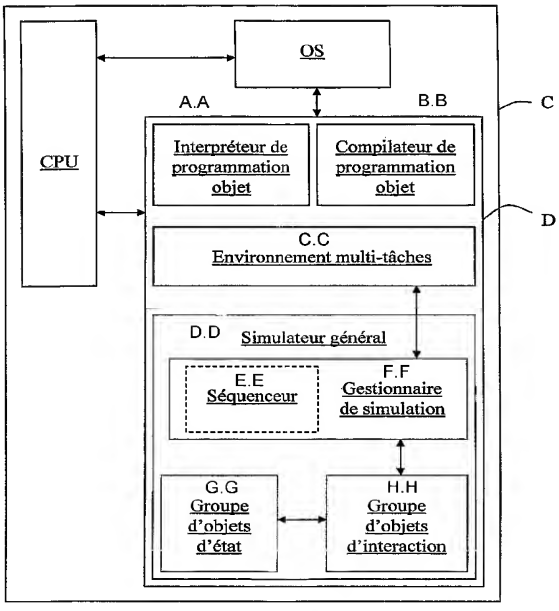
(75) Inventeurs/Déposants *(pour US seulement)* : **TISSEAU, Jacques** [FR/FR]; 3, rue Aristide Briand, F-29200 Brest (FR). **HARROUET, Fabrice** [FR/FR]; Milin Ar Pont, F-29290 Guipronvel (FR). **REDOU, Pascal** [FR/FR]; 4, rue des Lavoirs, F-29600 Morlaix (FR). **KERDELO, Sébastien** [FR/FR]; 35, rue Maria Chapdeleine, F-29200 Brest (FR). **CHARLES, Christian** [FR/FR]; 19, boulevard de la Corniche, F-29217 Plougonvelin (FR).

(74) Mandataire : **PLACAIS, Jean-Yves**; Cabinet Netter, 36, avenue Hoche, F-75008 Paris (FR).

[Suite sur la page suivante]

(54) Title: DEVICE FOR SIMULATION OF THE REAL WORLD BY ASYNCHRONOUS AND CHAOTIC PROCESSING

(54) Titre : DISPOSITIF DE SIMULATION DU MONDE RÉEL PAR TRAITEMENT ASYNCHRONE ET CHAOTIQUE



A.A... OBJECT PROGRAMMING INTERPRETER
B.B... OBJECT PROGRAMMING COMPILER
C.C... MULTI-TASK ENVIRONMENT
D.D... GENERAL SIMULATOR
E.E... SEQUENCER
F.F... SIMULATION CONTROLLER
G.G... STATE OBJECTS GROUP
H.H... INTERACTION OBJECTS GROUP

(57) Abstract: A computer (C) is embodied to permit programming by active objects in multi-task mode, which are representative of systems for simulation and comprises a device (D) for real world simulation. The device comprises an object simulation programme for combined evolution of some at least of the active objects, comprising i) objects of state each containing at least one item of spatial and/or temporal and/or property data defining a current state, ii) interaction objects each containing the definition of at least one of the state objects and at least one function applicable to at least one of said state objects and defining at each point in time the simulated system topology and iii) a simulation manager which may operate by sequences on a selection of interaction objects and to activate each interaction object only once for each sequence, according to an order varying in an at least partially random manner from one sequence to the other such as to apply each of the functions thereof to the actual state of each state object designated thereby for the evolution of the state thereof towards a new state.

(57) Abrégé : Un ordinateur (C) est agencé de manière à supporter en mode multi-tâches une programmation par objets activés, représentatifs de systèmes à simuler, et héberge un dispositif (D) de simulation du monde réel. Ce dispositif comprend un logiciel de simulation par objets de l'évolution conjointe de certains au moins des objets activés comportant i) des objets d'état contenant chacun au moins une donnée d'espace et/ou de temps et/ou au moins une donnée de propriété, définissant un état courant, ii) des objets d'interaction contenant chacun la désignation d'au moins l'un des objets d'état et d'au moins une fonction applicable à l'un au moins de ces objets d'état, et définissant à chaque instant la topologie du système simulé, et iii) un gestionnaire de simulation capable de travailler

[Suite sur la page suivante]

WO 2005/081141 A2

(81) États désignés *(sauf indication contraire, pour tout titre de protection nationale disponible)* : AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BW, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NA, NI, NO, NZ, OM, PG, PH, PL, PT, RO, RU, SC, SD, SE, SG, SK, SL, SY, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, YU, ZA, ZM, ZW.

(84) États désignés *(sauf indication contraire, pour tout titre de protection régionale disponible)* : ARIPO (BW, GH, GM, KE, LS, MW, MZ, NA, SD, SL, SZ, TZ, UG, ZM, ZW), eurasien (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), européen (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HU, IE, IS, IT, LT, LU, MC, NL, PL, PT, RO, SE, SI, SK, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Publiée :

— *sans rapport de recherche internationale, sera republiée dès réception de ce rapport*

En ce qui concerne les codes à deux lettres et autres abréviations, se référer aux "Notes explicatives relatives aux codes et abréviations" figurant au début de chaque numéro ordinaire de la Gazette du PCT.

par séquences sur une sélection d'objets d'interaction, et d'activer chaque objet d'interaction une unique fois lors de chaque séquence, selon un ordre variant de façon au moins partiellement aléatoire d'une séquence à l'autre, de manière à appliquer chacune de ses fonctions à l'état courant de chaque objet d'état qu'il désigne pour faire évoluer son état vers un nouvel état courant.

DISPOSITIF DE SIMULATION DU MONDE RÉEL PAR TRAITEMENT ASYNCHRONE ET CHAOTIQUE

L'invention concerne la simulation informatique du monde réel, et notamment de l'évolution temporelle de milieux objets de phénomènes physiques et/ou chimiques et/ou biologiques.

Comme le sait l'homme de l'art, de très nombreux objets (animés ou inanimés) ou substances interagissent avec les milieux (ou media) dans lesquels ils sont placés. Or, lorsque l'homme conçoit un nouvel objet, comme par exemple un bateau, ou bien lorsqu'il tente de comprendre un mécanisme réactionnel, comme par exemple le développement d'un cancer, il lui est généralement impossible de prendre en compte toutes les interactions, ou du moins les principales, notamment lorsqu'il utilise des maquettes. Par conséquent, il arrive fréquemment que l'objet final ne procure pas les résultats escomptés, ou que l'explication avancée ne corresponde pas à la réalité, en raison de la complexité et/ou de la combinaison des interactions présentes dans le monde réel. Ainsi, un bateau peut s'avérer incapable de supporter certains états de mer, alors même qu'il avait fait l'objet d'études préalables poussées, y compris dans un bassin de carène.

Afin de mieux anticiper le comportement final d'un objet ou produit, ou de mieux comprendre un mécanisme réactionnel, de nombreux industriels et de nombreux chercheurs utilisent des dispositifs de simulation informatique du monde réel.

Ces dispositifs de simulation reposent généralement sur une modélisation des phénomènes physiques et/ou chimiques et/ou biologiques sous la forme d'équations différentielles. En l'absence de solution analytique, la simulation informatique doit s'appuyer sur des méthodes itératives, qui passent d'un état courant du système considéré à un état suivant de ce même système. Chaque système est généralement décomposé en sous-systèmes ou mailles dont les évolutions respectives sont calculées en parallèle, de façon synchrone. Par conséquent, l'état du système à l'instant T+1 résulte de l'application en parallèle des phénomènes sélectionnés à l'état de chaque maille à l'instant T, sans se préoccuper des conséquences que cela peut induire entre mailles. En outre, cette technique de simulation ne favorise pas les modifica-

tions des paramètres de simulation en cours de travail, ni la combinaison de modèles de natures différentes, ce qui nuit à sa généralité d'application.

L'invention a donc pour but d'améliorer la situation.

Elle propose à cet effet un dispositif de simulation du monde réel, propre à être implanté dans un ordinateur capable de supporter en mode multi-tâches une programmation par objets activés, représentatifs de systèmes à simuler.

Ce dispositif se caractérise par le fait qu'il comprend un logiciel de simulation par objets de l'évolution conjointe de certains au moins des objets activés, comportant :
- des objets d'état contenant chacun au moins une donnée d'espace et/ou de temps et/ou au moins une donnée de propriété, définissant un état courant,
- des objets d'interaction contenant chacun la désignation d'au moins l'un des objets d'état et d'au moins une fonction applicable à l'un au moins de ces objets d'état, et définissant à chaque instant la topologie du système simulé,
- un gestionnaire de simulation capable de travailler par séquences sur une sélection d'objets d'interaction, et d'activer chaque objet d'interaction une unique fois lors de chaque séquence, selon un ordre variant de façon au moins partiellement aléatoire d'une séquence à l'autre, de manière à appliquer chacune de ses fonctions à l'état courant de chaque objet d'état qu'il désigne pour faire évoluer son état vers un nouvel état courant.

En d'autres termes, le dispositif selon l'invention fonctionne selon un mode asynchrone, du fait que les états respectifs des objets activés varient les uns après les autres au sein de chaque séquence, compte tenu des états respectifs des autres objets activés, et chaotique, du fait que l'ordre de traitement de chaque objet activé varie de façon aléatoire d'une séquence à l'autre. Il est possible d'activer ou supprimer à tout moment (c'est-à-dire en temps réel) un ou plusieurs objets, afin de modifier les conditions de travail et/ou le système simulé sans qu'il faille recommencer intégralement la simulation, ce qui confère au dispositif un véritable caractère interactif.

Le logiciel de simulation peut en outre comprendre des objets d'interaction interne contenant la désignation d'un seul objet d'état et d'au moins une fonction applicable à cet objet d'état,

et des objets d'interaction mutuelle contenant la désignation d'au moins deux objets d'état et d'au moins une fonction applicable à des données de propriété de ces objets d'état désignés.

Par ailleurs, certains au moins des objets d'état peuvent comprendre une donnée de propriété qui représente une grandeur intensive, et/ou certains au moins des objets d'interaction peuvent comporter une fonction faisant intervenir une grandeur extensive ou intensive.

On peut également utiliser des objets d'état qui pour certains au moins d'entre eux, comprennent des sous-objets d'état, ainsi qu'éventuellement des sous-objets d'interaction opérant sur ces sous-objets d'état.

En outre, le logiciel de simulation peut comporter des classes d'objets qui définissent des structures d'objets d'état et d'objets d'interaction. Les objets d'états et d'interaction sont alors dérivés de ces classes par instanciation.

De plus, le logiciel de simulation peut comporter un ordonnanceur capable de fonctionner soit selon un mode en temps réel, dans lequel il fonctionne selon une fréquence choisie, soit selon un mode en temps virtuel, dans lequel il fonctionne de façon périodique mais pendant des durées variables d'une période à l'autre.

D'autres caractéristiques et avantages de l'invention apparaîtront à l'examen de la description détaillée ci-après, et des dessins annexés, sur lesquels :

- la figure 1 illustre de façon très schématique, sous la forme de blocs fonctionnels, un ordinateur équipé d'un exemple de réalisation d'un dispositif de simulation selon l'invention, et
- la figure 2 est un organigramme détaillant un exemple de fonctionnement d'un dispositif selon l'invention.

Les dessins annexés pourront non seulement servir à compléter l'invention, mais aussi contribuer à sa définition, le cas échéant.

L'invention concerne un dispositif de simulation du monde réel, interactif.

Comme cela est illustré schématiquement et de façon fonctionnelle sur la figure 1, un dispositif de simulation selon l'invention D peut-être installé dans un ordinateur C comportant un système d'exploitation OS et des moyens de traitement et de calcul CPU adaptés à un fonctionnement dans un mode multi-tâches, tel que celui offert par l'environnement oRis décrit notamment dans le document "Systèmes multi-agents", pages 499 à 524, RSTI - TSI, 21/2002. Un tel environnement multi-tâches est particulièrement bien adapté à la programmation par objets activés, par exemple en langage C++ ou Java. L'environnement multi-tâches oRis est couplé, comme le dispositif D qui est illustré, à un compilateur (ici nommé "compilateur de programmation objet"). Par ailleurs, l'environnement oRis peut-être couplé, comme le dispositif D qui est illustré, à un traducteur en langage C++ (ici nommé "interpréteur de programmation objet") afin d'améliorer son efficacité par compilation. Cet interpréteur peut même être adapté de manière à constituer un compilateur en ligne dans lequel le code exécuté est un code compilé en ligne et modifiable de façon dynamique. Un tel environnement multi-tâches, constituant une évolution d'oRis, est connu sous le nom ARéVi.

Le dispositif D comprend un logiciel de simulation de l'évolution conjointe d'objets activés (ici nommé "simulateur général"). Plus précisément, ce logiciel de simulation (ou simulateur général) comporte des objets qui peuvent être décomposés en deux groupes.

Un premier groupe (nommé "groupe d'objets d'état") comprend des objets dits "d'état" contenant chacun une ou plusieurs données d'espace (X,Y,Z) et/ou de temps (T) et/ou une ou plusieurs données de propriété (ou paramètres, ou encore attributs), qui définissent ensemble un état courant pour l'objet d'état concerné.

Dans le cas le plus simple, dans lequel il n'existe qu'un seul objet d'état activé, l'objet d'interaction peut ne pas désigner l'objet d'état, celui-ci étant alors désigné de façon implicite.

Par ailleurs, dans le cas le plus simple, la donnée d'espace-temps d'un objet d'état peut se réduire au seul temps.

Certains au moins des objets d'état comprennent une donnée de propriété qui représente une grandeur intensive.

Dans un premier exemple relatif à un réacteur chimique, un objet d'état peut-être un volume non localisé, et une donnée de propriété peut-être une concentration d'une substance chimique.

Dans un second exemple relatif à une veine sanguine, un objet d'état peut-être une maille volumique localisée dans l'espace, une donnée d'espace peut-être la localisation géométrique dans un repère choisi, et une donnée de propriété peut-être une concentration chimique ou la température à l'intérieur d'une maille.

Un second groupe (nommé "groupe d'objets d'interaction") comprend des objets dits "d'interaction" (ou activités) contenant chacun la désignation d'au moins l'un des objets d'état et d'au moins une fonction applicable à l'un au moins de ces objets d'état désignés. Le mot "fonction" doit être compris dans sa définition informatique et non mathématique, selon laquelle elle prend quelque chose (n'importe quoi de défini) pour délivrer un résultat. Une fonction peut ainsi désigner une méthode, ou un procédé, ou encore un comportement à mettre en oeuvre.

Il est important de noter qu'une fonction peut dépendre des données de propriété d'objets d'état.

Certains au moins des objets d'interaction comprennent une donnée de propriété qui représente une grandeur intensive ou extensive.

Dans le premier exemple précité, relatif à un réacteur chimique, un objet d'interaction peut-être une réaction chimique qui manipule les données de propriété "concentration(s)" de l'objet d'état "volume", et la réaction chimique peut-être caractérisée par un paramètre "vitesse de réaction" qui peut dépendre des données de propriété "concentration(s)".

Dans le second exemple précité, relatif à la veine sanguine, un objet d'interaction peut-être un "diffuseur thermique" qui équilibre les données de propriété de température entre deux

mailles adjacentes, et le diffuseur thermique peut-être caractérisé par le paramètre "conductivité thermique".

Chaque objet d'interaction (ou activité) est donc associé à au moins un objet d'état par une fonction, et un objet d'état peut être associé à plusieurs objets d'interaction (ou activités). Une fonction ne modifie que les paramètres du ou des objets d'état auxquels elle est associée, sans modifier les paramètres des autres objets d'état et objets d'interaction. Par ailleurs, les caractéristiques des objets d'état peuvent permettre de choisir la fonction associée aux objets d'interaction ou d'en modifier les caractéristiques. En outre, une fonction peut-être modifiée de façon temporaire ou permanente en fonction des paramètres d'un objet d'état associé.

Comme on le verra plus loin, un objet d'interaction peut constituer une interface définie, généralement avec une orientation, entre deux milieux dits "source" et "cible" (éventuellement confondus) constituant des objets d'état. L'interface fait alors le lien entre les deux milieux, qui sont en liaison à un instant donné (notion de synchronicité forte), et modifie leurs paramètres respectifs (par exemple, l'interface peut être une différence de densité entre deux milieux qui provoque une diffusion moléculaire à une certaine vitesse).

Tout type d'interface peut être envisagé, et notamment des interfaces de transport (par exemple de molécules, de chaleur, de courant, et analogues), et des interfaces de relaxation (par exemple de réactions chimiques ou nucléaires). Par conséquent, un milieu peut ne pas avoir d'extension spatiale, celle-ci étant induite, tout comme les éventuelles vitesses, par une ou plusieurs interfaces. Par ailleurs, une activité appliquée à un milieu peut modifier le milieu, comme par exemple des phénomènes de relaxation, ce qui correspond à une auto-activation interne au milieu.

Le logiciel de simulation (ou simulateur général) comprend également un gestionnaire de simulation couplé aux groupes d'objets d'état et d'objets d'interaction et agencé de manière à créer son propre séquenceur ou ordonnanceur (ou en anglais "*scheduler*") afin de travailler de façon séquentielle sur une sélection d'objets d'interaction dudit groupe d'objets d'interaction. Plus précisément, le gestionnaire de simulation est chargé d'activer une unique fois lors de chaque séquence, sous le contrôle du séquenceur (ou scheduler) qu'il crée pour

l'occasion, chaque objet d'interaction sélectionné, selon un ordre qui varie de façon au moins partiellement aléatoire d'une séquence à l'autre, afin d'appliquer chacune de ses fonctions à l'état courant de chaque objet d'état qu'il désigne de manière à faire évoluer son état vers un nouvel état courant.

En d'autres termes, comme cela est illustré à titre d'exemple sur l'organigramme de la figure 2, l'utilisateur choisit tout d'abord, dans une étape 10, parmi les premier et second groupes d'objets, un ou plusieurs objets d'état et un ou plusieurs objets d'interaction qui se rapportent à ce ou ces objets d'état choisis, afin que le dispositif D simule l'évolution spatio-temporelle du système représenté par lesdits objets choisis. Cela "pré-active" chaque objet d'interaction choisi au sein du logiciel de simulation.

Puis, dans une étape 20, le gestionnaire de simulation initialise un compteur de séquences en plaçant la valeur n du compteur à 1, et crée une liste d'objets d'état et d'objets d'interaction. La première séquence (n=1) commence par une étape 30 dans laquelle le gestionnaire de simulation sélectionne de façon aléatoire, au sein de la liste d'objets d'interaction, l'un des objets d'interaction choisis, ce qui l'active momentanément. Chaque fonction de l'objet d'interaction sélectionné et activé est alors appliquée à l'état courant de chaque objet d'état de la liste d'objets d'état avec lequel il est en relation, ce qui modifie éventuellement son état en cours. L'activation et l'application (ou exécution) de chaque fonction à chaque objet d'état choisi constituent l'étape 40.

Puis, dans une étape 50, le gestionnaire de simulation supprime de la liste d'objets d'interaction de la séquence en cours l'objet d'interaction qu'il vient d'appliquer.

Ensuite, dans une étape 60, le gestionnaire de simulation effectue un test destiné à déterminer s'il reste d'autres objets d'interaction à appliquer dans la liste d'objets d'interaction de la séquence en cours.

Si la liste d'objets d'interaction de la séquence en cours n'est pas vide, le gestionnaire de simulation retourne à l'étape 30 afin de procéder à la sélection de façon aléatoire d'un objet d'interaction restant. Comme indiqué ci-avant, il active alors ce nouvel objet d'interaction sélectionné et applique chacune de ses fonctions à l'état courant de chaque objet d'état,

éventuellement modifié par l'activation de l'objet d'interaction précédent (lequel ne peut plus être utilisé dans la séquence en cours). Le gestionnaire de simulation reproduit ces opérations (sélection-activation, application et mise(s) à jour) autant de fois qu'il y a d'objets d'interaction sélectionnés dans sa liste d'objets d'interaction, de sorte que chaque fonction de chaque objet d'interaction soit appliquée une unique fois à chaque objet d'état activé auquel il est associé. Une fois cela terminé, la première séquence (n=1), correspondant à l'instant T=0, est achevée.

Lorsque la liste d'objets d'interaction de la séquence en cours est vide, le gestionnaire de simulation incrémente d'une unité, dans une étape 70, la valeur en cours n du compteur de séquences. Bien entendu, cette étape 70 comporte un test sur le nombre de séquences à effectuer. Si le nombre de séquences effectuées est égal au nombre maximal prévu, le gestionnaire de simulation met fin à la simulation. En revanche, s'il reste au moins une séquence à effectuer, le gestionnaire de simulation retourne à l'étape 20 pour effectuer une nouvelle séquence correspondant à un instant T+1, T+2, ..., T+n. Il réitère alors à chaque nouvelle séquence les opérations précitées.

La durée de la simulation, et donc le nombre maximal de séquences effectuées par le gestionnaire de simulation, dépend de l'application concernée, ou du paramétrage choisi par l'utilisateur compte tenu de l'application. Mais, la simulation peut être interrompue à chaque instant par l'utilisateur à l'aide d'une instruction d'arrêt transmise au logiciel de simulation grâce à une interface homme/machine de l'ordinateur C. Il est important de noter qu'une simulation interrompue à la demande d'un utilisateur peut-être reprise ultérieurement.

En raison de ce mode de fonctionnement séquentiel et chaotique du logiciel de simulation, l'utilisateur peut à chaque instant intervenir dans une simulation, soit sous la forme d'un "avatar" pour interagir lui-même avec l'objet de la simulation, par exemple le milieu, soit pour ajouter à, ou supprimer de, sa sélection un ou plusieurs objets d'état et/ou un ou plusieurs objets d'interaction. L'utilisateur peut également décider de modifier au moins partiellement la définition (ou structure) d'un ou plusieurs objets d'état ou d'interaction. Cela confère au logiciel de simulation une grande interactivité.

Trois exemples permettant de mieux comprendre ce que représentent les objets d'état et d'interactions sont donnés ci-après. Pour éviter d'alourdir inutilement la description, les exemples sont simplifiés et réduits à ce qui est nécessaire pour permettre à l'homme du métier de comprendre comment on peut mettre en oeuvre l'invention.

Un premier exemple concerne la simulation de la cinétique chimique au sein d'un réacteur chimique. Le réacteur chimique constitue un objet d'état représentant un milieu comportant des substances chimiques, et au sein duquel peuvent survenir N réactions chimiques constituant autant d'objets d'interaction.

L'objet d'état réacteur chimique est par exemple associé à des données de propriété représentant les concentrations (C1(T), C2(T), ..., Cn(T)) à l'instant T des n substances chimiques présentes initialement dans le réacteur. Ces concentrations définissent ici l'état (à l'instant T) de l'objet d'état réacteur chimique, dont on souhaite simuler l'évolution temporelle.

Chaque objet d'interaction réaction chimique est par exemple caractérisé par des données de propriété représentant, d'une première part, la vitesse de la réaction V = f(C1, C2, ..., Cn), d'une deuxième part, les réactifs concernés par la réaction (R1, R2, ..., Rn), d'une troisième part, les produits résultants de la réaction (P1, P2, ..., Pm), et d'une quatrième part les coefficients stoechiométriques définissant les proportions initiales des réactifs à T=0. En outre, chaque objet d'interaction réaction chimique est par exemple associé à une fonction représentant le comportement de la réaction : dans un premier temps on lit les concentrations (C1(T), C2(T), ..., Cn(T)) en cours des réactifs présents à l'instant T dans le réacteur chimique, dans un deuxième temps on calcule la vitesse de réaction V qui dépend des concentrations des réactifs présents dans le réacteur chimique, et dans un troisième temps on modifie les concentrations des réactifs et des produits après un temps dT à la vitesse de réaction V.

Ici chaque objet d'interaction n'agit que sur un unique objet d'état, de sorte qu'il constitue un objet d'interaction interne.

Dans ce premier exemple, l'utilisateur va donc choisir son objet d'état (c'est-à-dire la composition initiale de son réacteur chimique) et ses N objets d'interaction (c'est-à-dire les N réactions chimiques qui sont impliquées du fait de la composition choisie). Le gestionnaire de simulation peut alors débuter son traitement séquentiel.

Comme indiqué précédemment, il débute une première séquence en sélectionnant de façon aléatoire l'un des N objets d'interaction choisis. Cet objet d'interaction est activé afin qu'il applique sa fonction (comportement de la réaction) à l'état courant de l'unique objet d'état, défini ici par les concentrations (C1(T), C2(T), ..., Cn(T)) en cours des réactifs (lors de la première séquence T=0). Le gestionnaire met à jour les concentrations (C1(T), C2(T), ..., Cn(T)) des réactifs. Puis, il sélectionne de façon aléatoire l'un des N-1 objets d'interaction restants, afin de l'activer et d'appliquer sa fonction au nouvel état courant de l'unique objet d'état. Il reproduit ces opérations N fois de sorte que chaque fonction de chaque objet d'interaction soit appliquée une unique fois à l'objet d'état. La première séquence est alors terminée.

Le gestionnaire débute alors une nouvelle séquence, correspondant à l'instant T+1, si cela s'avère nécessaire, en réitérant les opérations précitées. La simulation prend fin lorsqu'il ne reste plus de réactifs pour produire des produits, ou bien lorsque l'utilisateur adresse une instruction d'arrêt au logiciel de simulation.

Un deuxième exemple concerne la simulation de la diffusion moléculaire de M diffuseurs au sein d'un espace découpé en K mailles.

Chacune des K mailles constitue un objet d'état caractérisé, par exemple, par une donnée de position (ou localisation topographique ou géographique dans un repère 3D) et par des données de propriété représentant les concentrations (C1(T), C2(T), ..., Cn(T)) à l'instant T de n substances chimiques (S1, S2, ..., Sn) présentes dans la maille. Ces concentrations définissent ici l'état (à l'instant T) de la maille.

Chacun des M diffuseurs constitue un objet d'interaction caractérisé, par exemple, par des données de propriété (ou paramètres) représentant, d'une première part, la substance diffusée Si, d'une deuxième part, la vitesse de diffusion tridimensionnelle V = (Vx, Vy, Vz) de la

substance Si, et d'une troisième part, les deux mailles entre lesquelles s'effectue la diffusion. En outre, chaque diffuseur (objet d'interaction) est par exemple associé à une fonction représentant son comportement : dans un premier temps on lit la concentration Ci(T) de la substance Si à diffuser dans chacune des deux mailles, dans un deuxième temps on calcule les quantités à diffuser (par exemple en utilisant la loi de Fick généralisée), et dans un troisième temps on modifie la concentration du diffuseur Si dans chacune des deux mailles après un temps dT à la vitesse de diffusion V. Les données de propriété représentent ici les attributs (ou paramètres) de la fonction (comportement du diffuseur).

Ici chaque objet d'interaction agit sur deux objets d'état, de sorte qu'il constitue un objet d'interaction mutuelle. Par ailleurs, chaque objet d'interaction présente ici une structure (ou définition) classique reposant sur au moins une fonction désignant au moins un objet d'état, ainsi qu'éventuellement des paramètres ou règle(s), ou encore loi(s), liés à la fonction et constituant des données de propriété. Mais, leur structure peut être plus complexe, par exemple lorsqu'elle comporte un ou plusieurs sous-objets d'interaction. En outre, les mailles sont ici de type tridimensionnel (3D), mais dans certaines applications elles peuvent être de type bidimensionnel (2D), voire même unidimensionnel (1D).

Dans ce deuxième exemple, l'utilisateur va donc choisir ses K objets d'état (c'est-à-dire les K mailles de diffusion) et ses M objets d'interaction (c'est-à-dire les M diffuseurs). Le gestionnaire de simulation peut alors débuter son traitement séquentiel.

Il débute une première séquence en sélectionnant de façon aléatoire l'un des M objets d'interaction choisis. Cet objet d'interaction est activé afin qu'il applique sa fonction (comportement du diffuseur), compte tenu de ses propres attributs, à l'état courant des K objets d'état (mailles), défini ici par les concentrations (C1(T), C2(T), ..., Cn(T)) en cours des substances chimiques (S1, S2, ..., Sn) dans les différentes mailles (lors de la première séquence T=0). Le gestionnaire met à jour les concentrations (C1(T), C2(T), ..., Cn(T)) des substances (S1, S2, ..., Sn) dans chacune des K mailles. Puis, il sélectionne de façon aléatoire l'un des N-1 objets d'interaction restants, afin de l'activer et d'appliquer sa fonction au nouvel état courant de chaque objet d'état. Il reproduit ces opérations N fois de sorte que chaque fonction de chaque objet d'interaction soit appliquée une unique fois à chaque objet d'état. La première séquence est alors terminée.

Le gestionnaire débute alors une nouvelle séquence, correspondant à l'instant T+1, si cela s'avère nécessaire, en réitérant les opérations précitées. La simulation prend fin lorsque les concentrations (C1(T), C2(T), ..., Cn(T)) des substances chimiques (S1, S2, ..., Sn) sont respectivement identiques dans chacune des K mailles, ou bien lorsque l'utilisateur adresse une instruction d'arrêt au logiciel de simulation.

Dans un troisième exemple, on peut combiner les deux exemples précédents de manière à simuler l'évolution de la cinétique chimique et de la diffusion moléculaire au sein d'une même veine.

Par exemple, on choisit K objets d'état constituant N réacteurs chimiques localisés dans l'espace 3D. Chaque réacteur chimique est par exemple associé à une donnée de position (ou localisation géographique dans un repère 3D) et à des données de propriété représentant les concentrations (C1(T), C2(T), ..., Cn(T)) à l'instant T de n substances chimiques (S1, S2, ..., Sn) présentes dans la maille. Ces concentrations définissent ici l'état (à l'instant T) de la maille.

Par ailleurs, on choisit par exemple N+M objets d'interaction constituant N réactions chimiques et M diffuseurs.

Chaque réaction chimique est par exemple associée à des données de propriété représentant, d'une première part, la vitesse de la réaction V = f(C1, C2, ..., Cn), d'une deuxième part, les réactifs concernés par la réaction (R1, R2, ..., Rn), à T=0, d'une troisième part, les produits résultants de la réaction (P1, P2, ..., Pm), et d'une quatrième part les coefficients stoechiométriques définissant les proportions initiales des réactifs à T=0. Chaque réaction chimique est également, par exemple, associé à une fonction représentant le comportement de la réaction : dans un premier temps on lit les concentrations (C1(T), C2(T), ..., Cn(T)) en cours des réactifs présents à l'instant T dans le réacteur chimique, dans un deuxième temps on calcule la vitesse de réaction V qui dépend des concentrations des réactifs présents dans le réacteur chimique, et dans un troisième temps on modifie les concentrations des réactifs et des produits après un temps dT à la vitesse de réaction V.

Chaque diffuseur est par exemple associé à des données de propriété représentant, d'une première part, la substance diffusée Si, d'une deuxième part, la vitesse de diffusion tridimensionnelle V = (Vx, Vy, Vz) de la substance Si, et d'une troisième part, les deux mailles entre lesquelles s'effectue la diffusion. Chaque diffuseur est également, par exemple, associé à une fonction représentant son comportement : dans un premier temps on lit la concentration Ci(T) de la substance Si à diffuser dans chacune des deux mailles, dans un deuxième temps on calcule les quantités à diffuser (par exemple en utilisant la loi de Fick généralisée), et dans un troisième temps on modifie la concentration du diffuseur Si dans chacune des deux mailles après un temps dT à la vitesse de diffusion V. Les données de propriété représentent ici les attributs (ou paramètres) de la fonction (comportement du diffuseur).

Le fonctionnement du dispositif est alors identique à celui présenté dans les exemples précédents, mais cette fois les concentrations des différentes substances évoluent non seulement en fonction des réactions chimiques, mais également en fonction des mailles où elles s'effectuent. Par conséquent, chaque séquence comporte N+M tirages aléatoires permettant d'appliquer successivement les fonctions des N+M objets d'interaction aux K réacteurs chimiques.

Bien entendu, l'invention peut s'appliquer à des applications beaucoup plus complexes que celles présentées ci-dessus à titre d'exemple illustratif. Elle s'applique également à des applications plus simples dans lesquelles le logiciel n'intervient que sur un unique objet d'état activé, à l'aide d'un unique objet d'interaction interne. Par ailleurs, dans certaines applications l'un au moins des objets d'état peut présenter une structure (ou définition) complexe reposant sur un ou plusieurs sous-objets d'état (présentant la structure classique d'un objet d'état), éventuellement associé à un ou plusieurs sous-objets d'interaction (présentant la structure classique d'un objet d'interaction).

Il est maintenant fait référence au tableau de l'annexe X1, pour une description, au niveau du code, d'un exemple de mise en oeuvre de l'invention faisant intervenir un objet d'état appelé "milieu", et plus précisément "classe milieu", et un objet d'interaction appelé "interface", et plus précisément "classe interface". Le but de cette application est d'associer une indication de couleur à une indication de densité au sein du milieu.

Le code n'apparaît que dans la seconde colonne du tableau, dont la première colonne ne comporte que des repères alphanumériques ne faisant pas partie du code.

L'annexe X1 commence, en A, par des déclarations "*#include*" qu'il est inutile de détailler du fait qu'elles sont bien connues de l'homme du métier. Ensuite, en B, est déclarée une classe *interface,* associée à la classe *milieu* traitée en détail en C. Après l'en-tête de classe C1, la classe *milieu* comprend la déclaration de méthodes spécifiques C20 à C30, puis en C4 une fonction (ou méthode) "*activate*", et enfin des variables protégées énoncées en C5. Dans l'annexe X1, les déclarations de méthodes (ou fonctions) sont allégées des définitions de type/paramètres, puisque l'homme de métier retrouvera celles-ci dans l'énoncé détaillé de chaque méthode.

Parmi les variables protégées indiquées en C5, on peut noter :
- des "*_interfaces*", qui sont regroupées au sein d'un tableau d'un type particulier dit StlVector (classe standard C++), et
- une variable "*_it*", qui est de type ArRef (classe propre à l'environnement ARéVi).

Vient ensuite le détail de la classe *interface* (par souci d'homogénéité elle reste désignée par B). Elle comprend une méthode de définition et une méthode de construction, suivies de déclaration de méthodes spécifiques B20 à B31 et de la méthode *activate* B4. Enfin sont définies des variables protégées B5 où l'on note à nouveau en particulier :
- une "*_source*" qui est de type *milieu*, et
- une "*_target*" qui est également de type *milieu*.

Les définitions des méthodes de la classe *milieu* interviennent ensuite (en C11-C14 et C20-C30) en grand détail. Les rubriques C11 à C13 définissent le constructeur de la classe *milieu*. En C11 sont initialisées les variables protégées de la classe milieu.

En C12 apparaît une définition d'une forme 3D dite "Vrml". Le code Vrml permet de faire de la visualisation 3D en ligne. Cette forme est définie par une chaîne de caractères conformément à la syntaxe Vrml décrite par exemple à l'adresse http://www.vrml.org. Cette forme est ensuite visualisée et mise en couleurs.

La rubrique C13 associe au *milieu* une activité destinée à déclencher la méthode *activate* de la rubrique C14.

La rubrique C14 est un destructeur de l'objet d'état *milieu.*

Les rubriques C20 à C30 incluse donnent le détail de la construction des méthodes spécifiques déjà évoquées sous la même identité pour la classe *milieu.*

La méthode *activate* à appliquer au *milieu* est définie en C4. Elle permet de relier une indication de couleur à l'indication de densité, et plus précisément de changer la couleur par tirage aléatoire. En d'autres termes, on modifie les attributs rouge, vert, bleu du *milieu* chaque fois qu'*activate* est appliquée (ou activée).

Interviennent ensuite des définitions pour la classe *interface*, avec en B10-B11 la définition du constructeur. Plus précisément, en B10 on initialise les variables protégées de la classe interface, tandis qu'en B11 on associe à l'interface une activité destinée à appliquer la méthode activate définie en B4.

La rubrique B12 est un destructeur de l'objet d'interaction *interface*.

Viennent ensuite les détails des méthodes spécifiques B20 à B31 déjà citées.

La méthode *activate* de l'*interface* est définie en B4. Il s'agit de calculer une diffusivité en fonction d'une densité de *milieu source* et d'une densité de *milieu cible*, afin de propager l'écart entre les deux densités, si du moins cet écart est significatif. L'écart noté "*delta*" est calculé comme l'application d'une fonction diffusivité à la densité de *milieu source rhos*, diminuée de la densité de *milieu cible rhot*. Pour ce faire, on demande d'abord au *milieu source* et au *milieu cible* quelles sont leurs densités respectives. Puis, on calcule la valeur absolue des différences de couleurs. Si cette valeur absolue est supérieure à une valeur choisie (ici égale à 0,01), on augmente la densité la plus faible et on diminue la densité la plus forte. Puis, on effectue la mise à jour des valeurs à une vitesse choisie.

Les rubriques B60-B63 permettent de définir le type des fenêtres de visualisation 3D qui seront utilisées par l'application.

C'est alors qu'intervient un élément important pour l'invention qui est l'ordonnanceur ou séquenceur (ou en anglais "*scheduler*"). Les rubriques S10, S20, S21, S22, S30, S31 et S41 permettent d'initialiser la simulation.

En l'espèce, la rubrique S10 initialise un *scheduler* en temps virtuel, c'est-à-dire que son fonctionnement n'est pas astreint à respecter le temps réel, mais chacune de ses itérations représente logiquement, et non physiquement, une durée de une milliseconde (1 ms). Bien entendu, le *scheduler* peut également fonctionner en temps réel. Dans ce cas, chacune de ses itérations dure physiquement une période choisie.

La rubrique S20 permet d'initialiser les dimensions du milieu grâce aux arguments de la ligne de commande. La rubrique S21 permet de créer et d'initialiser les milieux. La rubrique S22 permet de créer et d'initialiser les interfaces.

Les rubriques S31 et S32 réalisent un affichage auxiliaire, qui se fait ici par la voie spéciale de communication avec l'écran dit "flot d'erreur".

La rubrique S41 affiche la scène à traiter, dans des conditions de perspective choisies.

Enfin, la rubrique S50 est le programme principal *main* qui initialise le fonctionnement du système. Plus précisément, ce programme crée tout d'abord un système 3D, puis il l'appelle pour qu'il active la méthode donnée à la rubrique S10 afin qu'elle crée le scheduler (ou séquenceur), les milieux et l'interface.

On comprendra que la structure de classes, présentée ci-avant, peut être résumée de la manière indiquée schématiquement dans l'annexe X2.

La transition entre la représentation de l'annexe X2 et le code détaillé de l'annexe X1 est considérée comme accessible à l'homme du métier, dès lors qu'un exemple en a été donné.

L'invention trouve de très nombreuses applications dans de nombreux domaines techniques, et notamment dans les domaines de la chimie, de la pharmacie, de la physique, de l'aéronautique, de l'architecture, en particulier navale (par exemple pour l'étude comportemental d'un bateau ou d'une plate-forme offshore, en remplacement et/ou en complément des bassins de carène), de la médecine, notamment dans le cadre de l'étude du développement et du traitement de certaines maladies (par exemple les cancers) ou de mécanismes réactionnels (par exemple l'activation de l'insuline), de l'ergonomie, notamment pour la réalisation d'équipements spécifiquement adaptés à des personnes handicapées, et dans le domaine de la circulation routière.

L'invention ne se limite pas aux modes de réalisation de dispositif de simulation décrits ci-avant, seulement à titre d'exemple, mais elle englobe toutes les variantes que pourra envisager l'homme de l'art dans le cadre des revendications ci-après.

Annexe X1

A	#include statements	
B	class Interface;	
C1	**class Milieu : public Object3D** { public: AR_CLASS(Milieu) AR_CONSTRUCTOR(Milieu)	
C20 C21	setDensity() getDensity()	
C22 C23	setColor() getColor()	
C24 C25	addSource(); removeSource();	
C26 C27 C28	addInterface() removeInterface() getNbInterfaces()	
C29 C30	getInterface() accessInterface()	
C4	**activate(ArRef<Activity> act, double dt);**	
C5	protected: StlVector<ArRef<Interface> > _interfaces; ArRef<VrmlMaterialInteractor> _it; double _red,_green,_blue; double _density; bool _changeDensity; double _isSource; };	
B1	**class Interface : public ArObject** { public: AR_CLASS(Interface) AR_CONSTRUCTOR(Interface)	

B20 B21	setVelocity() getVelocity()	
B22 B23	setDiffusivity() getDiffusivity()	
B24 B25 B26	setSource() getSource() accessSource()	
B27 B28 B29	setTarget() getTarget() accessTarget()	
B30 B31	getOtherEnd() accessOtherEnd()	
B4	**activate(ArRef<Activity> act, double dt);**	
B5	protected: ArRef<Milieu> _source; ArRef<Milieu> _target; double _velocity, _diffusivity; };	
	AR_CLASS_DEF(Milieu,Object3D)	
C11	Milieu::Milieu(ArCW & arCW) : Object3D(arCW), _interfaces(), _it(new_VrmlMaterialInteractor()), _red(1.0), _green(1.0), _blue(1.0), _density(0), _changeDensity(true), _isSource(false)	

C12	{ ArRef<VrmlShape3D> sh=new_VrmlShape3D(); sh->parseString(" \ Shape { \ appearance Appearance { \ material DEF MAT Material { \ diffuseColor 1.0 1.0 1.0 \ transparency 0.5 \ } \ } \ geometry Box { \ size 0.9 0.9 0.9 \ } \ }");	

C13	ArRef<Activity> act=new_Activity(0.1);
	act->setBehavior(thisRef(),&Milieu::activate);
	}	

C14	Milieu::~Milieu(void)
	{
	}	

C20	void
	Milieu::setDensity(double density)
	{
	_density = density;
	_changeDensity = true;
	}	

C21	void
	Milieu::getDensity(double & density)
	{
	density = _density;
	}	

C22	void
	Milieu::setColor(double red,
	double green,
	double blue)
	{
	_red=red;
	_green=green;
	_blue=blue;
	_it->writeDiffuseColorField(_red,_green,_blue);
	}	

C23	void
	Milieu::getColor(double & redOut,
	double & greenOut,
	double & blueOut) const
	{
	redOut=_red;
	greenOut=_green;
	blueOut=_blue;
	}	

C24	void
	Milieu::addSource(void)
	{
	_isSource = true;
	}	

C25	void
	Milieu::removeSource(void)
	{
	_isSource = false;
	}	

C26	void
	Milieu::addInterface(ArRef<Interface> interface)
	{
	_interfaces.push_back(interface);
	}	

C27	void
	Milieu::removeInterface(ArRef<Interface> interface)
	{
	for(unsigned int i=_interfaces.size();i--;)
	{
	if(_interfaces[i]==interface)
	{
	StlVectorFastErase(_interfaces,i);
	break;
	}
	}
	}	

C28	unsigned int
	Milieu::getNbInterfaces(void) const
	{
	return(_interfaces.size());
	}	

C29	ArConstRef<Interface>
	Milieu::getInterface(unsigned int index) const
	{
	return(_interfaces[index]);
	}	

C30	ArRef<Interface>
	Milieu::accessInterface(unsigned int index)
	{
	return(_interfaces[index]);
	}	

C4	bool
	Milieu::activate(ArRef<Activity> act,
	double /* dt */)
	{
	if(_isSource) { _density = 1.0; }
	if(_changeDensity)
	{
	setColor(1.0,1.0-_density,1.0-_density);
	/*
	if(_density < 0.1) { setColor(1,1,1); }
	else { setColor(1,0,0); }
	*/
	_changeDensity = false;
	}
	return(true);
	}	

B10	AR_CLASS_DEF(Interface,ArObject) Interface::Interface(ArCW & arCW) : ArObject(arCW), _source(), _target(), _velocity(0), _diffusivity(0)	

B11	{
	ArRef<Activity> act=new_Activity(0.1);
	act->setBehavior(thisRef(),&Interface::activate);
	}	

B12	Interface::~Interface(void)
	{
	}	

B20	void
	Interface::setVelocity(double velocity)
	{
	_velocity = velocity;
	}	

B21	void
	Interface::getVelocity(double & velocity)
	{
	velocity = _velocity;
	}	

B22	void
	Interface::setDiffusivity(double diffusivity)
	{
	_diffusivity = diffusivity;
	}	

B23	void
	Interface::getDiffusivity(double & diffusivity)
	{
	diffusivity = _diffusivity;
	}	

B24	void
	Interface::setSource(ArRef<Milieu> source)
	{
	if(_source)
	{
	_source->removeInterface(thisRef());
	}
	_source=source;
	if(_source)
	{
	_source->addInterface(thisRef());
	}
	}	

B25	ArConstRef<Milieu>
	Interface::getSource(void) const
	{
	return(_source);
	}	

B26	ArRef<Milieu>
	Interface::accessSource(void)
	{
	return(_source);
	}	

B27	void
	Interface::setTarget(ArRef<Milieu> target)
	{
	if(_target)
	{
	_target->removeInterface(thisRef());
	}
	_target=target;
	if(_target)
	{
	_target->addInterface(thisRef());
	}
	}	

B28	ArConstRef<Milieu>
	Interface::getTarget(void) const
	{
	return(_target);
	}	

B29	ArRef<Milieu>
	Interface::accessTarget(void)
	{
	return(_target);
	}	

B30	ArConstRef<Milieu>
	Interface::getOtherEnd(ArConstRef<Milieu> m) const
	{
	return(m==_source ? _target : _source);
	}	

B31	ArRef<Milieu>
	Interface::accessOtherEnd(ArConstRef<Milieu> m)
	{
	return(m==_source ? _target : _source);
	}	

B4

```
bool
Interface::activate(ArRef<Activity> /* act */,
                    double /* dt */)
{
double rhos,rhot;
_source->getDensity(rhos);
_target->getDensity(rhot);
bool change=false;
if(fabs(rhot-rhos)>0.01)
 {
 double delta=_diffusivity*(rhos-rhot);
 rhos -= delta;
 rhot += delta;
 change=true;
 }
if(change)
 {
 _source->setDensity(rhos);
 _target->setDensity(rhot);
 }
return(true);
}
```

B60

```
class MyViewer : public Examiner3D
{
public:
 AR_CLASS(MyViewer)
 AR_CONSTRUCTOR(MyViewer)
protected:
 virtual void _onKeyPress(const Viewer3D::KeyPressEvent & evt);
};
```

B61

```
AR_CLASS_DEF(MyViewer,Examiner3D)

MyViewer::MyViewer(ArCW & arCW)
: Examiner3D(arCW)
{
}
```

B62

```
MyViewer::~MyViewer(void)
{
}
```

B63

```
void
MyViewer::_onKeyPress(const Viewer3D::KeyPressEvent & evt)
{
Examiner3D::_onKeyPress(evt);
if((evt.key=="+")||(evt.key=="-"))
  {
  double dist,yaw,pitch;
  computeCursorDirection(evt.xMouse,evt.yMouse,yaw,pitch);
  ArRef<Base3D> ray=new_Base3D();
  ray->setLocation(thisRef());
  ray->yaw(yaw);
  ray->pitch(pitch);
  ArRef<Object3D> found;
  getScene()->firstRayIntersection(ray,3,true,Milieu::CLASS(),
                                   found,dist,thisRef());
  if(found)
    {
    ArRef<Milieu> m(static_cast<Milieu*>(found.c_ptr()));
    cerr << "Found " << m->getAlias() << "  " << evt.key << endl;
    if(evt.key=="+") { m->addSource(); }
    if(evt.key=="-") { m->removeSource(); }
    }
  }
}
```

S10

```
ArRef<Scheduler>
simulationInit(void)
{
ArRef<Scheduler> sched=new_VirtualTimeScheduler(1e-3);

unsigned int nbX=40,nbY=20,nbZ=2;
```

S20	if(ArSystem::getCommandLine().size()>1)
	{
	strToUint(nbX,ArSystem::getCommandLine()[1]);
	}
	if(ArSystem::getCommandLine().size()>2)
	{
	strToUint(nbY,ArSystem::getCommandLine()[2]);
	}
	if(ArSystem::getCommandLine().size()>3)
	{
	strToUint(nbZ,ArSystem::getCommandLine()[3]);
	}	

S21	unsigned int x,y,z;
	StlVector<StlVector<StlVector<ArRef<Milieu> > > > t;
	for(x=0;x<nbX;x++)
	{
	StlVector<StlVector<ArRef<Milieu> > > ex;
	t.push_back(ex);
	for(y=0;y<nbY;y++)
	{
	StlVector<ArRef<Milieu> > ey;
	t.back().push_back(ey);
	for(z=0;z<nbZ;z++)
	{
	ArRef<Milieu> m=new_Milieu();
	t.back().back().push_back(m);
	m->setPosition(x,y,z);
	}
	}
	}	

S22

```
for(x=0;x<nbX;x++)
 {
 for(y=0;y<nbY;y++)
  {
  for(z=0;z<nbZ;z++)
   {
   ArRef<Interface> i;
   double diffusivity = 0.5;
   //double velocity = 1.0;
   if(x>0)
    {
    i=new_Interface();
    i->setSource(t[x][y][z]);
    i->setTarget(t[x-1][y][z]);
        i->setDiffusivity(diffusivity);
        i->setVelocity(0.0);
    }
   if(y>0)
    {
    i=new_Interface();
    i->setSource(t[x][y][z]);
    i->setTarget(t[x][y-1][z]);
        i->setDiffusivity(diffusivity);
        i->setVelocity(0.0);
    }
   if(z>0)
    {
    i=new_Interface();
    i->setSource(t[x][y][z]);
    i->setTarget(t[x][y][z-1]);
        i->setDiffusivity(diffusivity/1000);
        i->setVelocity(0.0);
    }
   }
  }
 }
```

S31

```
cerr << ArClass::find("Interface")->getNbInstances(true)
   << " interfaces" << endl;
```

S32	cerr << ArClass::find("Milieu")->getNbInstances(true) << " milieux" << endl;	

S41	ArRef<Viewer3D> v=NEW_MyViewer();
	ArRef<BoundingBox3D> bbox=new_BoundingBox3D();
	v->getScene()->computeBoundingBox(bbox);
	v->setFarDistance(5.0*bbox->getMaxSize());
	v->setNearDistance(0.001*v->getFarDistance());
	unsigned int w,h;
	Viewer3D::getScreenSize(w,h);
	v->setWindowGeometry(w-400,0,400,400);
	v->setMapped(true);
	
	return(sched);
	}	

S50	int
	main(int argc,
	  char ** argv)
	{
	ArSystem3D init(argc,argv);
	return(init.simulate(&simulationInit));
	}	

5

Annexe X2

Représentation simplifiée

Classe Interface	Constructeur d'objets
Méthodes spécifiques	setVelocity() getVelocity() setDiffusivity() getDiffusivity() setSource() getSource() accessSource() setTarget() getTarget() accessTarget() getOtherEnd() accessOtherEnd()
Méthode activate	activate()

Classe Milieu	Constructeur d'objets
Méthodes spécifiques	setDensity() getDensity() setColor() getColor() addSource() removeSource() addInterface() removeInterface() getNbInterface() getInterface() accessInterface()
Méthode activate	activate()

REVENDICATIONS

1. Dispositif (D) de simulation du monde réel, propre à être implanté dans un ordinateur (C) agencé pour supporter en mode multi-tâches une programmation par objets activés, représentatifs de systèmes à simuler, caractérisé en ce qu'il comprend un logiciel de simulation par objets de l'évolution conjointe de certains au moins desdits objets activés, comportant :

- des objets d'état contenant chacun au moins une donnée d'espace et/ou de temps et/ou au moins une donnée de propriété, définissant un état courant,
- des objets d'interaction contenant chacun la désignation d'au moins l'un desdits objets d'état et d'au moins une fonction applicable à l'un au moins de ces objets d'état, et définissant à chaque instant la topologie du système simulé,
- un gestionnaire de simulation capable de travailler par séquences sur une sélection d'objets d'interaction, et à activer chaque objet d'interaction une unique fois lors de chaque séquence, selon un ordre variant de façon au moins partiellement aléatoire d'une séquence à l'autre, de manière à appliquer chacune de ses fonctions à l'état courant de chaque objet d'état qu'il désigne pour faire évoluer son état vers un nouvel état courant.

2. Dispositif selon la revendication 1, caractérisé en ce que ledit logiciel de simulation comprend des objets d'interaction interne, aptes à contenir chacun la désignation d'un seul objet d'état et d'au moins une fonction applicable à cet objet d'état, et des objets d'interaction mutuelle, aptes à contenir chacun la désignation d'au moins deux objets d'état et d'au moins une fonction applicable à des données de propriété de ces objets d'état désignés.

3. Dispositif selon l'une des revendications 1 et 2, caractérisé en ce que ledit logiciel de simulation est agencé pour modifier certaines au moins desdites fonctions en fonction d'au moins une donnée de propriété d'au moins un objet d'état associé.

4. Dispositif selon l'une des revendications 1 à 3, caractérisé en ce que ledit logiciel de simulation est agencé pour sélectionner certaines au moins desdites fonctions en fonction d'au moins une donnée de propriété d'au moins un objet d'état associé.

5. Dispositif selon l'une des revendications 1 à 4, caractérisé en ce que certains au moins des objets d'état comprennent une donnée de propriété représentant une grandeur intensive.

6. Dispositif selon l'une des revendications 1 à 5, caractérisé en ce que certains au moins des objets d'interaction ont une fonction faisant intervenir une grandeur extensive ou intensive.

7. Dispositif selon l'une des revendications précédentes, caractérisé en ce que certains au moins des objets d'état comprennent des sous-objets d'état.

8. Dispositif selon la revendication 7, caractérisé en ce que certains au moins des objets d'état comprennent des sous-objets d'interaction opérant sur lesdits sous-objets d'état.

9. Dispositif selon l'une des revendications précédentes, caractérisé en ce que ledit logiciel de simulation comporte des classes d'objets définissant des structures d'objets d'état et d'objets d'interaction, lesdits objets d'états et d'interaction étant dérivés de ces classes par instanciation.

10. Dispositif selon l'une des revendications précédentes, caractérisé en ce que ledit logiciel de simulation comporte un ordonnanceur propre à fonctionner selon l'un de deux modes choisis parmi un mode en temps réel, dans lequel il fonctionne selon une fréquence choisie, et un mode en temps virtuel, dans lequel il fonctionne de façon périodique mais pendant des durées variables d'une période à l'autre.

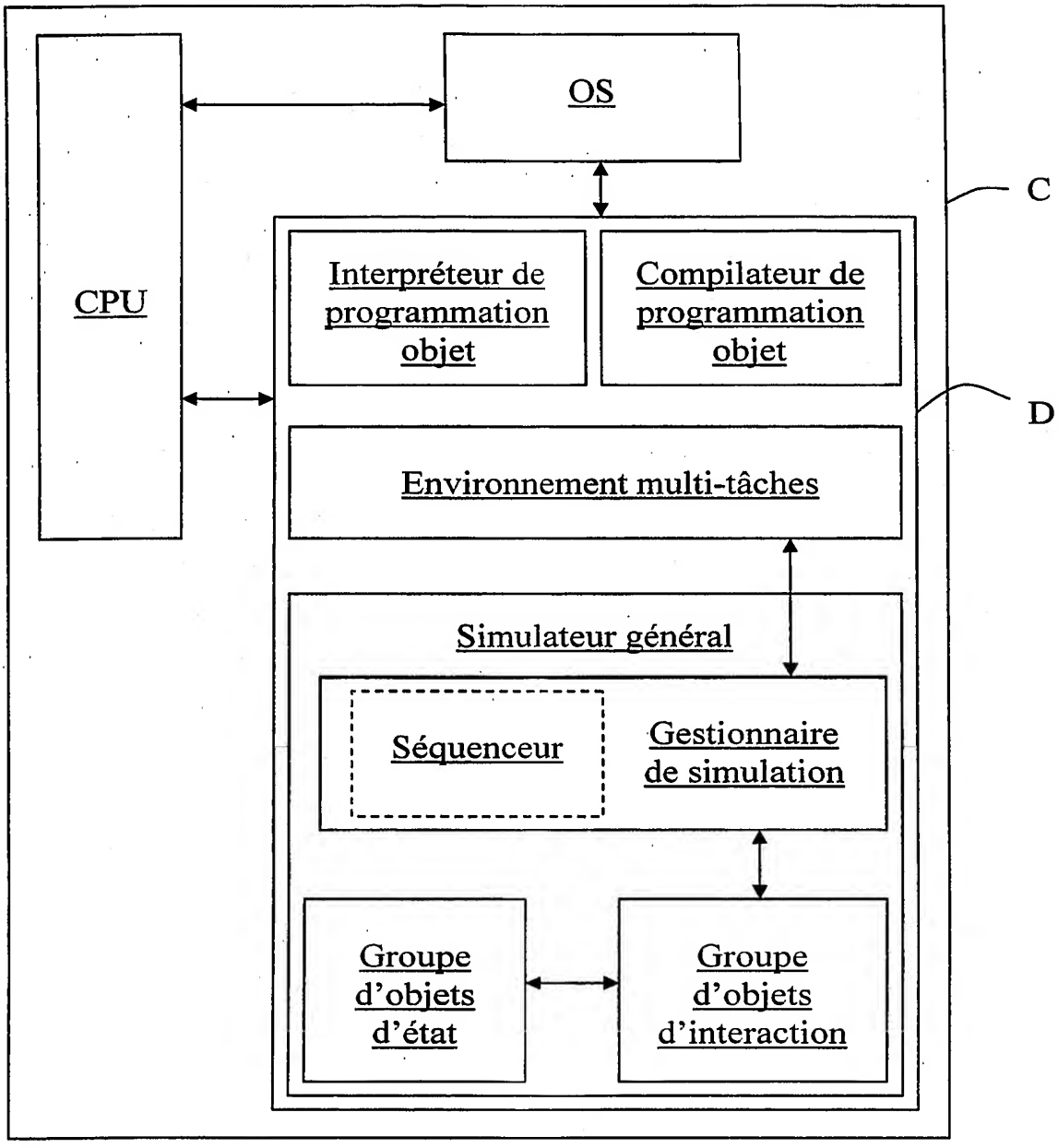

OS
CPU
C
Interpréteur de programmation objet
Compilateur de programmation objet
D
Environnement multi-tâches
Simulateur général
Séquenceur
Gestionnaire de simulation
Groupe d'objets d'état
Groupe d'objets d'interaction

FIG.1

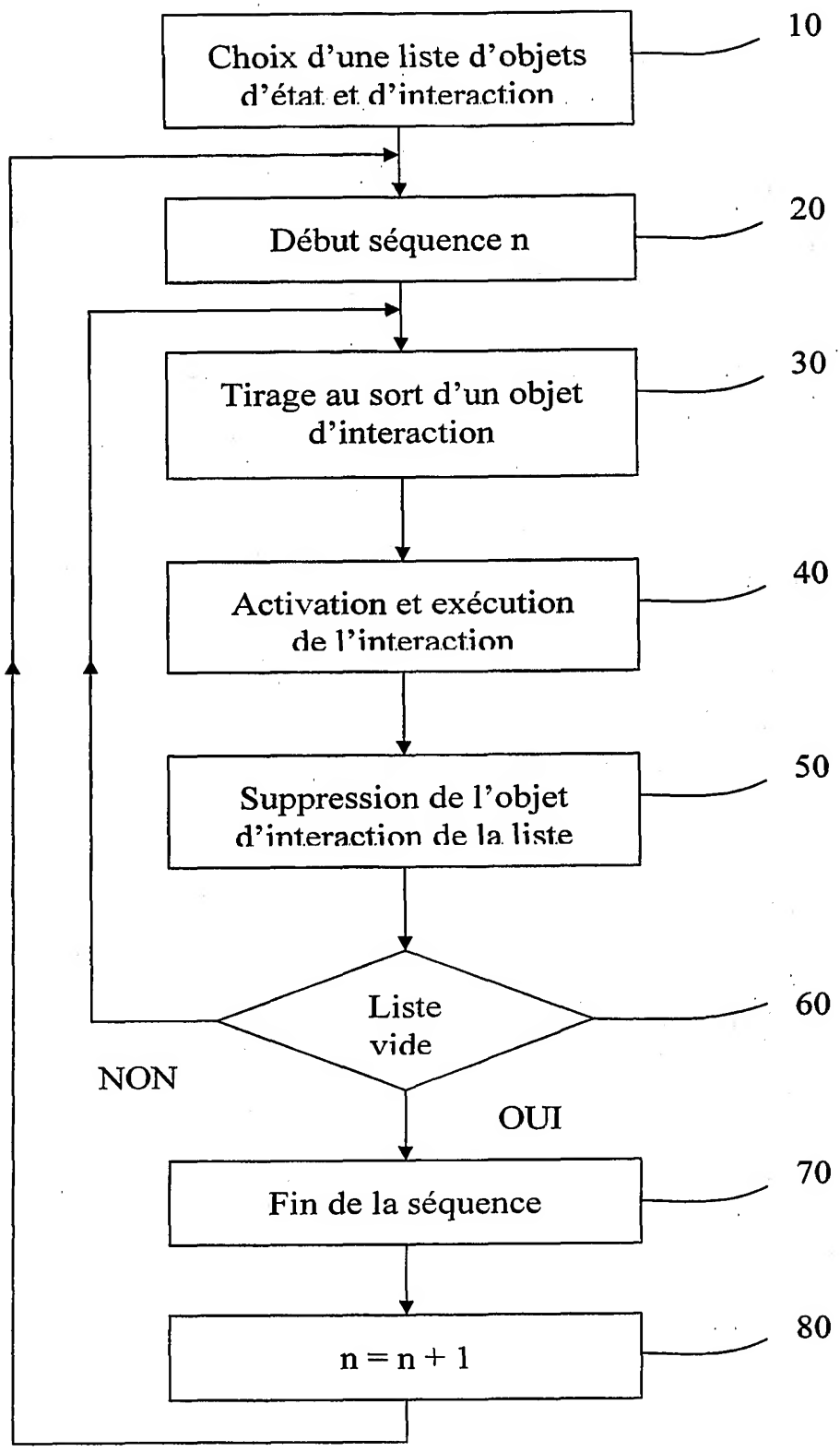

Choix d'une liste d'objets d'état et d'interaction
10
Début séquence n
20
Tirage au sort d'un objet d'interaction
30
Activation et exécution de l'interaction
40
Suppression de l'objet d'interaction de la liste
50
Liste vide
60
NON
OUI
Fin de la séquence
70
n = n + 1
80

FIG.2